

MINT: a Computer Program for Adaptive Monte Carlo Integration and Generation of Unweighted Distributions*

Paolo Nason

INFN, Sezione di Milano-Bicocca, Piazza della Scienza 3, 20126 Milan, Italy
E-mail: Paolo.Nason@mib.infn.it

ABSTRACT: In this note I illustrate the program MINT, a FORTRAN program for Monte Carlo adaptive integration and generation of unweighted distributions.

*This document has been produced using $\text{\TeX}_{\text{MACS}}$ (see <http://www.texmacs.org>)

Contents

1. Introduction	1
2. The algorithm	2
3. Folded integration	3
4. Generation of p-tuples x^1, \dots, x^q.	4
5. The code	5

1. Introduction

The purpose of this note is to illustrate the program MINT, a FORTRAN program for Monte Carlo adaptive integration and generation of unweighted distributions. The program performs the following task: given a function $f(x^1, \dots, x^n)$, defined in the unit n -dimensional cube, and such that the function

$$\tilde{f}(x^1, \dots, x^p) = \int_0^1 dx^{p+1} \int_0^1 dx^{p+2} \dots \int_0^1 dx^n f(x^1, \dots, x^n) \quad (1.1)$$

is positive for x^1, \dots, x^p in the p -dimensional unit cube, it computes the integral of f , and generates x^1, \dots, x^p points in the unit cube, distributed with probability

$$\tilde{f}(x^1, \dots, x^p) dx^1 \dots dx^p. \quad (1.2)$$

The case $p = n$ is contemplated, i.e. MINT can also perform the task of unweighting a positive distribution.

A popular program to perform adaptive Monte Carlo multi-dimensional integration is the VEGAS program [1]. It uses an importance-sampling method, where the sampling rate is independent for each coordinate. The method works well for factorizable singularities, and it has the advantage that the sampling information one needs to store grows only linearly with the number of dimensions.

A common problem encountered in particle physics phenomenology, besides the integration of a given multidimensional function, is the generation of its arguments with a probability proportional to its value. The SPRING-BASES program [2] use the VEGAS algorithm to perform the integration of a positive function, and then can generate events distributed with a probability proportional to the integrand. It first stores the integration result and the maximum value of the function for each cell of the adaptive mesh found by VEGAS. In the generation stage, a cell is chosen with a probability proportional to the

corresponding value of the integral, and then a point in the cell is generated using the hit and miss technique. This method is highly efficient, but it has the disadvantage that the amount of storage space it requires grows exponentially with the dimension.

MINT is a replacement for the SPRING-BASES package. It differs from it in the following way. In order to generate the phase-space point, it does not store the value of the integral for each cell. It only stores an upper bound of the value of the function in each cell. The multidimensional stepwise function that equals the upper bound of the function to be integrated in each cell is in fact an upper bound for the whole function, and it is easy to generate phase space points distributed according to it. Using again the hit and miss technique, we can then generate points according to the original distribution. In order to save space, MINT uses an upper bounding functions that is the products of a set of step-wise functions, each of them associated with a coordinate. The storage requirement for such a function grows only linearly with the dimension. The ability to deal with non-positive functions is achieved in MINT by *folding over* (in a sense that will be specified later) the integrand along the directions $p + 1, \dots, n$.

The problem of unweighting a distribution of the form (1.1) arises in the context of the method of refs. [3, 4, 5] for the inclusion of next-to-leading corrections in Monte Carlo generated events.

2. The algorithm

The algorithm is in essence the VEGAS algorithm. It is implemented in MINT in the following way. Assume that we deal with the n -dimensional integral of a function $f(x^1, \dots, x^n) \geq 0$ in the unit hypercube. We divide the $[0, 1]$ interval for each coordinate in m bins of variable length,

$$x_{l-1}^k \leq x^k \leq x_l^k \text{ for } l = 1, \dots, m \text{ and } k = 1, \dots, n.$$

We then define n monotonic, continuous functions $h^k(y^k)$, with $0 < y^k < 1$ and $k = 1, \dots, n$, such that

$$h^k\left(\frac{l}{m}\right) = x_l^k \text{ for } l = 0, \dots, m, \quad (2.1)$$

and linear (i.e. having constant first derivative) in all the intervals $(l-1)/m < y < l/m$.

We have

$$\int f(x) d^n x = \int f(h(y)) \prod_{k=1}^n \frac{dh^k(y^k)}{dy^k} dy^k. \quad (2.2)$$

Observe that, given y^k , if l is the bin where y lies (i.e. $(l-1)/m < y < l/m$), we have

$$\frac{dh^k(y^k)}{dy^k} = (x_l^k - x_{l-1}^k) \times m. \quad (2.3)$$

We begin the adaptation process with $x_l^k = l/m$ for all $k = 1, \dots, n$, $l = 1, \dots, m$. We would like to find optimal h^k functions, such that the integration process has small errors. This is done as follows. We perform several iteration of the integration. In each iteration

we generate a set \mathcal{I}_N of N random points y in the unit hypercube $0 \leq y^k \leq 1$, $k = 1, \dots, n$, and estimate the integral with the formula

$$I = \frac{1}{N} \sum_{y \in \mathcal{I}_N} f(h(y)) \prod_{k=1}^n \frac{dh^k(y^k)}{dy^k}. \quad (2.4)$$

To each y we associate a point $x = h(y)$, and we call l^k the bin where x^k lies. We accumulate the value of f and the number of hits in two arrays R_l^k and N_l^k , with $k = 1, \dots, n$ and $l = 1, \dots, m$

$$R_l^k = \sum_{y \in \mathcal{I}_N} \theta(l-1 \leq y^k < l) \times f(x) \prod_{k'=1}^n (x_{l^k}^{k'} - x_{l^k-1}^{k'}), N_l^k = \sum_{y \in \mathcal{I}_N} \theta(l-1 \leq y^k < l), \quad (2.5)$$

where

$$\prod_{k=1}^n (x_{l^k}^k - x_{l^k-1}^k)$$

is the volume of the cell where x lies. After all the N points have been generated, we define for all $k = 1, \dots, n$ and $l = 0, \dots, m$

$$I_l^k = \sum_{j=1}^l \frac{R_j^k}{N_j^k}, I_0^k = 0 \quad (2.6)$$

and a corresponding continuous piecewise linear function $i^k(x^k)$ such that $i^k(x_l^k) = I_l^k$ for $l = 0, \dots, m$. Notice that $i^k(\bar{x}^k)$ is an estimate of the integral of f in the $n-1$ dimensional hyper-rectangle defined by the condition $0 \leq x^k \leq \bar{x}^k$. We now find the new x_l^k points by solving the equation

$$i^k(x_l^k) = i^k(1) \frac{l}{m} \quad (2.7)$$

and then we go to the next iteration. It is clear that when all the R_j^k/N_j^k are equal, the procedure has reached stability, the x_l^k no longer change, and we are probing regions of equal importance with (in the average) the same number of points. Furthermore, this procedure is nearly optimal if

$$f(x^1, \dots, x^n) = f_1(x^1) \times \dots \times f_n(x^n). \quad (2.8)$$

3. Folded integration

If the given function f is not positive definite, but its integral over a subset of the integration variables is positive, we can turn it into a positive function by folding it over itself a sufficient number of times in the given subset of the integration variables. More precisely, we do the following. Suppose we want to fold each k coordinate p_k times. We define the function

$$\bar{f}(z^1, \dots, z^n) = \frac{1}{p_1} \sum_{l_1=0}^{p_1-1} \dots \frac{1}{p_n} \sum_{l_n=0}^{p_n-1} \left[f(h(y)) \prod_{k=1}^n \frac{dh^k(y^k)}{dy^k} \right]_{y=y(z,l)}, \quad (3.1)$$

where

$$y^k(z^k, l^k) = \frac{l_k + z^k}{p_k}$$

so that, as the variable z^k spans the $[0, 1]$ interval, y^k spans each of the p_k equal subintervals

$$\frac{l_k}{p_k} < y^k < \frac{l_k + 1}{p_k}. \quad (3.2)$$

It is clear that

$$\int f(x) d^n x = \int f(h(y)) \prod_{k=1}^n \frac{dh^k(y^k)}{dy^k} dy^k = \int \bar{f}(z^1, \dots, z^n) dz^1 \dots dz^n. \quad (3.3)$$

It is convenient for practical purposes to choose the p_k among the integer divisors of m .

4. Generation of p -tuples x^1, \dots, x^q .

In order to generate p -tuples x^1, \dots, x^p , with $p \leq n$, and a probability distribution

$$P(x^1, \dots, x^p) = \begin{cases} \frac{\int f(x) dx^{p+1} \dots dx^n}{\int f(x) d^n x} & \text{for } p < n \\ \frac{f(x)}{\int f(x) d^n x} & \text{for } p = n \end{cases}, \quad (4.1)$$

we generate the whole n -tuple, and then discard the $p+1, \dots, n$ coordinates. We can use an arbitrary amount of folding on the $p+1, \dots, n$ coordinates, while we must keep the $1, \dots, p$ coordinates unfolded (i.e. $p^1, \dots, p^q = 1$). We then proceed with the generation of the z variables using the \bar{f} function, with probability

$$P(z^1, \dots, z^n) = \frac{\bar{f}(z^1, \dots, z^n)}{\int \bar{f}(z^1, \dots, z^n) dz^1 \dots dz^n}. \quad (4.2)$$

We seek for \bar{f} an upper bound of the form

$$\bar{f}(z^1, \dots, z^n) \leq u^1(z_1) \times \dots \times u^n(z_n), \quad (4.3)$$

where $u^k(z^k)$ are stepwise functions in the $0 < z^k < 1$ interval divided into m/p_k equal subintervals. The $u^k(z^k)$ are initialized according to

$$u^k(z^k) = \left[\int \bar{f}(z^1, \dots, z^n) dz^1 \dots dz^n \right]^{1/n}. \quad (4.4)$$

We perform a large number of calls to the function \bar{f} , at (uniform) random values of its argument. If the bound is violated

$$\bar{f}(z^1, \dots, z^n) \geq u^1(z_1) \times \dots \times u^n(z_n), \quad (4.5)$$

each of the $u^k(z_k)$ is increased by a fixed factor f in the subinterval containing z_k . The value

$$f = 1 + \frac{1}{10n} \quad (4.6)$$

is found to work reasonably well. After a sufficiently large number of calls, the u will stabilize.

In practice, in the MINT program, the upper bounding envelope is computed during the folded integration.

5. The code

The code is available at the URL

```
http://moby.mib.infn.it/~nason/POWHEG/FNOpaper/mint-integrator.f .
```

It is composed by two user-callable routines, `mint` and `gen`. Furthermore, the common block

```
integer ifold(ndimmax)
common/cifold/ifold
```

must be set, specifying how many times each dimension is folded. The user function must have the form

```
function fun(x,w,ifl)
real * 8 fun,w,x(ndim)
```

When called with `ifl=0` it must return `fun=f(x)*w`. When called with `ifl=1` it returns `fun=f(x)*w`, but it may assume that, since the previous `ifl=0` call the variables that are not folded, and all the values that have been computed with them, have remain unchanged. The return value `f(x)*w` is *not* used by `mint` in this case. When called with `ifl=2`, `fun` must return the sum of all the return values up to (and including) the last `ifl=0` call. This apparently cumbersome procedure is needed to allow for enough flexibility, in order for a program to be able to compute the fraction of positive and negative contributions to the folded integral, which is needed in POWHEG applications.

To run the program, one first calls the subroutine

```
real * 8 xgrid(50,ndimmax),xint,ymax(50,ndimmax),ans,err
integer ndim,ncalls0,nitmax,imode
call mint(fun,ndim,ncalls0,nitmax,imode,xgrid,xint,ymax,ans,err)
c ndim: dimension of x in fun (ndim<=ndimmax)
c ncalls0: maximum number of calls per iteration
c nitmax: number of iterations
c imode: flag
```

- When called with `imode=0`, `mint` performs the integration of the absolute value of the function, finds the optimal grid `xgrid`, stores the answer in `xint` and `ans`, and the error in `err`.

- When called with `imode=1`, `mint` performs the folded integration. The grid is kept fixed at this stage. The array

```
integer ifold(ndimmax)
common/cifold/ifold
```

controls the folding, i.e. `ifold(k)` is the number of folds for `x(k)`. The number of folds must be an integer divisor of the number of bins, which is fixed to 50. The array `xgrid` and the value `xint` must have already been filled by a previous call to `mint` with `ifl=0`. The upper bounding envelope of the folded function is also computed at this stage, and stored in the array `ymax`.

The value and error of the integral are returned in `ans` and `error`. Once `ymax` has been setup in this way, one can call `gen` to generate events with the distribution of the folded function. Notice that negative values of the called function are not allowed at this stage.

One calls

```
imode=0
call gen(fun,ndim,xgrid,ymax,imode,x)
imode=1
do j=1,10000
call gen(fun,ndim,xgrid,ymax,imode,x)
...
enddo
imode=3
call gen(fun,ndim,xgrid,ymax,imode,x)
```

where the call with `imode=0` initializes the generation, `imode=1` generate the `ndim`-tuples, and `imode=3` prints out generation statistics. After a call to `gen` with `imode=1`, one can assume that the last call to the function `fun` was performed with the generated value of `x`, so that parameters depending upon `x` that are stored in common blocks by `fun` have consistent values.

References

- [1] G. Peter Lepage. Vegas: An Adaptive Multidimensional Integration Program. CLNS-80/447.
- [2] Setsuya Kawabata. A New Version of the Multidimensional Integration and Event Generation Package BASES/SPRING. *Comp. Phys. Commun.*, 88:309–326, 1995.
- [3] Paolo Nason. A New Method for Combining NLO QCD with Shower Monte Carlo Algorithms. *JHEP*, 11:040, 2004.
- [4] Paolo Nason and Giovanni Ridolfi. A Positive-Weight Next-to-Leading-Order Monte Carlo for Z Pair Hadroproduction. *JHEP*, 08:077, 2006.
- [5] Stefano Frixione, Paolo Nason, and Giovanni Ridolfi. A Positive-Weight Next-to-Leading-Order Monte Carlo for Heavy Flavour Hadroproduction, arXiv:0707.3088 [hep-ph].